Aalborg Universitet

AALBORG UNIVERSITY
DENMARK

Answering skyline queries over incomplete data with crowdsourcing (Extended Abstract)

Miao, Xiaoye; Gao, Yunjun; Guo, Su; Chen, Lu; Yin, Jianwei; Li, Qing

Published in:
Proceedings - 2020 IEEE 36th International Conference on Data Engineering, ICDE 2020

DOI (link to publication from Publisher):
10.1109/ICDE48307.2020.00235

Creative Commons License
CC BY 4.0

Publication date:
2020

Document Version
Accepted author manuscript, peer reviewed version

Link to publication from Aalborg University

Citation for published version (APA):
Miao, X., Gao, Y., Guo, S., Chen, L., Yin, J., & Li, Q. (2020). Answering skyline queries over incomplete data with crowdsourcing (Extended Abstract). In *Proceedings - 2020 IEEE 36th International Conference on Data Engineering, ICDE 2020* (pp. 2032-2033). [9101783] IEEE. Proceedings - International Conference on Data Engineering Vol. 2020-April https://doi.org/10.1109/ICDE48307.2020.00235

General rights
Copyright and moral rights for the publications made accessible in the public portal are retained by the authors and/or other copyright owners and it is a condition of accessing publications that users recognise and abide by the legal requirements associated with these rights.

- Users may download and print one copy of any publication from the public portal for the purpose of private study or research.
- You may not further distribute the material or use it for any profit-making activity or commercial gain
- You may freely distribute the URL identifying the publication in the public portal -

Take down policy
If you believe that this document breaches copyright please contact us at vbn@aub.aau.dk providing details, and we will remove access to the work immediately and investigate your claim.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TKDE.2019.2946798, IEEE Transactions on Knowledge and Data Engineering

Answering Skyline Queries over Incomplete Data with Crowdsourcing

Xiaoye Miao, Yunjun Gao, *Member, IEEE,* Su Guo, Lu Chen, Jianwei Yin, Qing Li, *Senior Member, IEEE*

Abstract—Due to the pervasiveness of incomplete data, incomplete data queries are vital in a large number of real-life scenarios. Current models and approaches for incomplete data queries mainly rely on the machine power. In this paper, we study the problem of *skyline queries over incomplete data with crowdsourcing*. We propose a novel query framework, termed as BayesCrowd, which takes into account the data correlation using Bayesian network. We leverage the typical *c-table* model on incomplete data to represent objects. Considering budget and latency constraints, we present a suite of effective task selection strategies. Moreover, we introduce a *marginal utility* function to measure the benefit of crowdsourcing one task. In particular, the probability computation of each object being an answer object is at least as hard as #SAT problem. To this end, we propose an *adaptive* DPLL (i.e., Davis-Putnam-Logemann-Loveland) algorithm to speed up the computation. Extensive experiments using both real and synthetic data sets confirm the superiority of BayesCrowd to the state-of-the-art method, in terms of execution time, monetary cost, and latency minimization.

Index Terms—Query Processing, Skyline Query, Incomplete Data, Crowdsourcing

1 Introduction

The skyline query finds the objects that are not dominated by any other object, where an object o dominates another object o' iff o is not worse than o' in all attributes, and is better than o' in at least one attribute. This query has a large application base in many real-life scenarios such as decision making, profiled based recommendation, location-based services [1], [2], [3], [4]. Take a dataset in movie recommendation system as an example. Each movie is represented by a vector containing ratings from audiences. For instance, for the three movies m_1 = (3, 2, 1), m_2 = (4, 2, 3), and m_3 = (2, 3, 2), each of them has ratings from three audiences where the higher the rating, the better. It is said, m_1 is dominated by m_2, as m_2 has equal/higher ratings to/than m_1 on three attributes. Thus, m_2 and m_3 are the skyline points.

In real-life applications, it is impossible for all audiences to watch/score a certain movie. Hence, some movie ratings are usually missing. As depicted in Table 1, there are five movies (i.e., objects) $\{o_1, o_2, o_3, o_4, o_5\}$ with ratings from five audiences (w.r.t. attributes) $\{a_1, a_2, a_3, a_4, a_5\}$. The movie/object o_2's value on the attribute a_2 is missing, and thus, it is denoted by the variable $Var(o_2, a_2)$. Obviously, the real answer objects of this skyline query cannot be obtained due to the data incompleteness.

Incomplete data are ubiquitous in a wide spectrum of real-life applications, owing to a variety of reasons such as instable sensor networks, data integration, data loss, privacy preservation, etc. For example, users tend to skip certain fields when they fill out on-line forms; participants choose to ignore some sensitive questions on surveys; publicly viewable satellite map services contain missing map data in many mobile applications; and in privacy-preserving applications, the data is incomplete deliberately in order to preserve the sensitivity of some attribute values. As a result, incomplete data queries have been extensively explored in the past decade, including skyline queries [5], [6], [7], [8], [9], [10], [11], [12], [13], [14], top-k queries [15], [16], similarity queries [17], [18], [19], and so forth. However, most of existing models and approaches for incomplete data queries only rely on the machine power.

TABLE 1
A Sample Dataset

ID	Name	Film Ratings from Audiences				
		a_1	a_2	a_3	a_4	a_5
o_1	Schindler's List (1993)	5	2	3	4	1
o_2	Se7en (1995)	6	Var(o_2, a_2)	2	2	2
o_3	The Godfather (1972)	1	1	Var(o_3, a_3)	5	3
o_4	The Lion King (1994)	4	3	1	2	1
o_5	Star Wars (1977)	5	Var(o_5, a_2)	Var(o_5, a_3)	Var(o_5, a_4)	1

As well known, the machine has limitations in some cases where the human is powerful [20]. Collective intelligence has become a hot topic, with the development of Web 3.0 and the emerging of artificial intelligence (AI) techniques. As a consequence, many crowdsourcing platforms [21] emerge, such as Amazon mechanical turk (AMT)[1], FigureEight[2], and Upwork[3], each of which acts as an intermediate between requesters and workers. On crowdsourcing platforms, a requester posts a series of tasks, and workers answer those tasks and get paid. Compared

- *X. Miao and J. Yin are with the Center for Data Science, Zhejiang University, 866 Yuhangtang Road, Hangzhou 310058, P. R. China. E-mail: {miaoxy, zjuyjw}@zju.edu.cn.*
- *Y. Gao (Corresponding Author), S. Guo, and J. Yin are with the College of Computer Science, Zhejiang University, 38 Zheda Road, Hangzhou 310027, P. R. China. E-mail: {gaoyj, guosu}@zju.edu.cn.*
- *L. Chen is with the Department of Computer Science, Aalborg University, Aalborg, Denmark. E-mail: luchen@cs.aau.dk.*
- *Q. Li is with the Department of Computing, the Hong Kong Polytechnic University, Hong Kong, P. R. China. E-mail: csqli@comp.polyu.edu.hk.*

Manuscript received XX XX, 2019; revised XX XX, 2019.

1. AMT is available at https://www.mturk.com/mturk/.
2. FigureEight is available at https://www.figure-eight.com/.
3. Upwork is available at https://www.upwork.com.

1041-4347 (c) 2019 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TKDE.2019.2946798, IEEE Transactions on Knowledge and Data Engineering

with conventional trade markets, the crowdsourcing platform offers a more free employment contract where workers can come and go as their wills freely. Towards this, in this work, we resort to *crowdsourcing* techniques to handle skyline queries over incomplete data.

The skyline problem with crowdsourcing relies on the definition of dominance relationship over *complete* data. It means that, the techniques and algorithms of skyline queries over incomplete data [5], [6], [7], [8], [9], [10], [11], [12], [13], [14] cannot be directly applied to our studied problem, since they mainly depend on the dominance relationship definition over *incomplete* data (as defined in [5]). In addition, the existing crowd skyline algorithms [22], [23] have the following shortcomings. The returned results may be inaccurate in [22], due to the *unary* questions to assess missing values of objects. In contrast, CrowdSky [23], as the state-of-the-art method, divides attributes into observed attributes and crowd ones, and all the values in crowd attributes are assumed missing. It is opposite of real scenarios where the values are usually missing over arbitrary attributes [24].

In this paper, we systematically study the problem of *skyline queries over incomplete data with crowdsourcing*. There are three key research challenges to solve the problem. (i) How do we capture the data correlation? (ii) How do we represent query result objects? (iii) What could we do to prioritize the crowd tasks? As long as these questions are answered satisfactorily, our proposed framework BayesCrowd will work well. To begin with, we train the Bayesian network over data attributes to capture the data correlation. Second, in BayesCrowd, we leverage a typical incomplete database representation system, i.e., *c-table* [25], which assigns a condition of becoming a query answer to every object. Hence, the object is not a query answer object if its condition is not satisfied. Finally, we consider budget and latency constraints, and develop a suite of effective task selection strategies. In brief, our key contributions in this paper are summarized as follows.

- We present a novel crowd skyline query framework BayesCrowd, which incorporates two main phases, i.e., the *modeling* phase and the *crowdsourcing* phase.
- In the modeling phase, we generate the condition of each object being a query answer object for *c-table* construction. In the crowdsourcing phase, we develop three task selection strategies in iteration policy under budget and latency constraints. The *marginal utility function* is introduced to quantify the benefit of crowdsourcing one task.
- As a key step of measuring the utility, the probability computation is at least as hard as the model counting problem (i.e., #SAT problem). We propose an adaptive DPLL (abbrev. of Davis-Putnam-Logemann-Loveland) (ADPLL for short) method to accelerate the computation.
- Extensive experimental evaluation using both real and synthetic data sets demonstrates that, our proposed framework BayesCrowd is superior to the state-of-the-art method in terms of both efficiency and accuracy, under a variety of parameter settings.

The rest of the paper is organized as follows. We describe some preliminaries in Section 2. We present the framework BayesCrowd in Section 3. Section 4 and Section 5 detail the c-table construction and the probability computation, respectively. Section 6 elaborates the crowd task selection strategies. The experimental results and our findings are reported in Section 7. The related work is reviewed in Section 8. Finally, we conclude our work with some directions for future work in Section 9.

TABLE 2
Symbols and Description

Notation	Description
$\mathcal{O}$	a set of objects
o_i	an object in $\mathcal{O}$
Q	a query over $\mathcal{O}$
$\mathcal{R}$	a query result set over $\mathcal{O}$
$\mathcal{C}$	the c-table of a query
$\phi(o)$	the condition of an object o in the c-table
$\mathrm{Var}(o_i, a_j)$	the variable representing the attribute a_j of an object o_i
$\Pr(\phi(o))$	the probability of the object o being a query answer

2 PRELIMINARIES

In this section, we introduce the skyline query and a typical representation model, i.e., *c-table*, for incomplete data. We formalize our studied problem. Table 2 summarizes the symbols used frequently in the rest of this paper.

Definition 1. **(Dominance relationship)**. Given two objects o_1 and o_2 from a dataset $\mathcal{O}$, o_1 dominates o_2 (denoted as $o_1 \prec o_2$) if and only if both of the following conditions hold: (i) o_1 is not worse than o_2 in any attribute, i.e., for each attribute i, $o_1.[i] \geq o_2.[i]$; and (ii) o_1 is better than o_2 in at least one attribute, i.e., $\exists j, o_1.[j] > o_2.[j]$.

For sake of clarity, $o_1.[i]$ denotes o_1's value in the attribute i, and we assume that, the larger the value, the better. Our solution likewise does work for the case of preferring smaller values. In this paper, we follow the definition above (which is generally employed on *complete* data), and assume that, there is no prior knowledge on the missing values. It indicates that, every missing value has non-zero probability of getting any value within the corresponding attribute domain, that is, there is no constraint on the possible values of missing values. Note that, our studied skyline problem considers both the *observed* dimensions and the *missing* information on incomplete data, which is reasonable and straightforward in practice. It is different from that designated for *incomplete* data, as defined in [5], which tests the comparable (observed) dimension(s) of each object pair, and *ignores* the missing information.

Definition 2. **(Skyline query)**. Given a dataset $\mathcal{O}$, a skyline query retrieves the objects, included in $\mathcal{R}$, which are not dominated by any other object in the dataset. Formally, $\mathcal{R} = \{o_i \mid \nexists o_j \in \mathcal{O}, o_j \prec o_i\}$.

The traditional skyline query was firstly introduced into the database community by Borzsony et al. [1]. Note that, the typical indices (such as R-tree family) and algorithms for skyline queries on complete data [2] are inapplicable due to the data incompleteness in the studied problem.

Definition 3. **(C-table)**. A *conditional database* (*c-table* for short), denoted as $\mathcal{C}$, is a group of object-condition pairs $\langle o \in \mathcal{O}, \phi(o) \rangle$, in which $\phi(o)$ is a propositional formula (called the object o's *condition*).

1041-4347 (c) 2019 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TKDE.2019.2946798, IEEE Transactions on Knowledge and Data Engineering

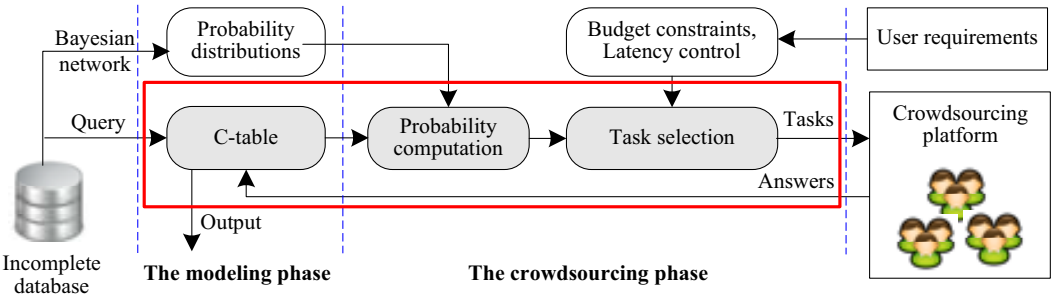


Fig. 1. The architecture of BayesCrowd

TABLE 3
The C-Table over the Sample Dataset

Object	Condition
o_1	$(\text{Var}(o_5, a_2) < 2) \vee (\text{Var}(o_5, a_3) < 3) \vee (\text{Var}(o_5, a_4) < 4)$
o_2	true
o_3	true
o_4	$(\text{Var}(o_2, a_2) < 3) \wedge [(\text{Var}(o_5, a_2) < 3) \vee (\text{Var}(o_5, a_3) < 1) \vee (\text{Var}(o_5, a_4) < 2)]$
o_5	$[(\text{Var}(o_5, a_2) > 2) \vee (\text{Var}(o_5, a_3) > 3) \vee (\text{Var}(o_5, a_4) > 4)] \wedge [(\text{Var}(o_5, a_2) > \text{Var}(o_2, a_2)) \vee (\text{Var}(o_5, a_3) > 2) \vee (\text{Var}(o_5, a_4) > 2)]$

The *c-table* model was firstly proposed by the foundational research work [25] in 1980s. In this paper, we utilize it to represent objects over incomplete dataset. Actually, the object's condition is a boolean combination of inequalities involving missing values (i.e., variables) and constants.

Example 1. For the skyline query on the sample dataset, the corresponding c-table is shown in Table 3. The conditions of o_2 and o_3 are true. It means that o_2 and o_3 are definitely the result objects of the skyline query. In addition, the condition of o_1, i.e., $\phi(o_1)$, is $(Var(o_5, a_2) < 2) \vee (Var(o_5, a_3) < 3) \vee (Var(o_5, a_4) < 4)$. It indicates that, if $\phi(o_1)$ is satisfied, then o_1 is a skyline result object. In other words, the probability of $\phi(o_1)$ (being satisfied) is equivalent to the possibility of the object o_1 being a skyline result object.

A crowd task in this paper is a triple choice (i.e., larger/smaller than, or equal to) to ask the relation of two operands in the inequality of a condition. It means that, each task corresponds to an inequality in the condition, we also call it an *expression*. For instance, for an expression "*Var*(o_5, a_2) < 2" in the c-table, the corresponding task is to ask "Is the variable *Var*(o_5, a_2) larger than, or smaller than, or equal to 2?" We will detail how to get the c-table (i.e., those conditions) in Section 4 later.

Definition 4. **(Our problem)**. Given an incomplete dataset $\mathcal{O}$ and a skyline query Q, the goal of our studied problem is to strategically select tasks $\mathcal{T}$ based on the c-table for crowdsourcing, so as to optimize crowd query accuracy with budget B and latency constraint L.

In addition, it is noteworthy that, uncertain/probabilistic data is a bit different from incomplete data. Uncertain data models the missing values on top of some probability density distributions. However, incomplete data does have zero prior knowledge on the missing values. As a result, most of skyline techniques/algorithms over uncertain data [26], [27], [28] cannot be applied to our studied problem.

Algorithm 1: BayesCrowd Framework

Input: an incomplete data set $\mathcal{O}$, a query Q
Output: the query result set $\mathcal{R}$
/* The modeling phase */
1: $\mathcal{C} \longleftarrow$ Get-CTable($\mathcal{O}, Q$)
/* The crowdsourcing phase */
2: select crowd tasks $\mathcal{T}$
3: post tasks $\mathcal{T}$ on the crowdsourcing platform
4: collect answers from crowd workers
5: derive the query result set $\mathcal{R}$
6: **return** $\mathcal{R}$

3 BayesCrowd Framework

In this section, we provide an overview of BayesCrowd, as depicted in Figure 1.

BayesCrowd mainly consists of two phases, i.e., *the modeling phase* and *the crowdsourcing phase*. In the modeling phase, the condition of each object being a query answer is generated using the c-table model, which makes preparation for the next phase. When entering the crowdsourcing phase, BayesCrowd takes into account users' requirements (e.g., budget constraints and latency control), and selects tasks for crowdsourcing, which involves a critical component, i.e., the probability computation. Then, the chosen tasks are posted to the crowdsourcing platform. The answers will be later returned by crowd workers. Finally, BayesCrowd outputs the query result set via updating the c-table.

Algorithm 1 shows the *general* procedure of BayesCrowd. It receives an incomplete dataset $\mathcal{O}$ (with learned probability distributions) and a query Q, and outputs the query result set $\mathcal{R}$. First, in the modeling phase, BayesCrowd constructs the c-table, denoted as $\mathcal{C}$, for the query Q via using the function Get-CTable, which will be explained in Section 4 (line 1). Then, in the second phase, BayesCrowd selects the awaiting crowd tasks, which needs multiple probability computation. Those chosen tasks are posted on crowdsourcing markets (lines 2-3). Note that, a suite of task selection strategies will be detailed in Section 6 later. Thereafter, it receives task answers from crowd workers, and infers the query result set $\mathcal{R}$ (lines 4-5). Finally, BayesCrowd returns $\mathcal{R}$, and terminates (line 6).

Prior to the modeling phase, there is a preprocessing step. In that step, BayesCrowd trains the Bayesian network over the dataset to capture the data correlation. Then, using the trained Bayesian network, it learns the probability distributions of missing values leveraging Bayes rules. In the implementation, the Bayesian network is trained using the

1041-4347 (c) 2019 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TKDE.2019.2946798, IEEE Transactions on Knowledge and Data Engineering

framework Banjo[4] for *structure learning* and the framework Infer.Net[5] for estimating network structure parameters. It is worth noting that, Bayesian network is more suitable to discrete values. For continuous values, we partition the whole domain into a series of value ranges (using some space partitioning techniques), and treat each range as a discrete value to process in BayesCrowd. Also, one can alternatively employ autoencoder architectures [29] to capture complex distributions. In addition, theoretically, the cleaner the dataset, the more accurate the learned probability distributions with Bayesian network. Hence, one could first clean the dataset (e.g., using some existing cleaning tools [30]) before training the Bayesian network, which is very common in practice. The problem of cleaning data is beyond the scope of this paper. Without loss of generality, it is assumed that the input dataset to BayesCrowd is clean.

4 C-Table Construction

In this section, we explain how to obtain the c-table efficiently. Note that, when objects are represented under the c-table model with *conditions*, the probability of an object o's condition, denoted as $\phi(o)$, being satisfied indicates the possibility of the object o being a query result object.

4.1 Deriving the Dominator Set

First, let us make an analysis. For the skyline query, an object o is a query answer if it is not dominated by any other object in the dataset. In other words, the condition of o being an answer object is to have at least one better attribute value for o than the objects that probably dominate o.

For an object $o \in \mathcal{O}$, let $\mathcal{D}(o)$ be the set of all the objects that are likely to dominate o on the incomplete dataset. For simplicity, we call it *the dominator set* of o throughout this paper. Then, the condition of o, i.e., $\phi(o)$, consists of $|\mathcal{D}(o)|$ conjuncts, and each of the conjuncts encompasses at most d disjuncts, where d is the number of data attributes. Without loss of generality, in the paper, the disjunct is usually called the expression or task, which is an inequality between a variable and a constant or between two variables. Formally, if the dominator set $\mathcal{D}(o)$ is $\{o_1, o_2, \cdots\}$, the condition of o, i.e., $\phi(o)$, can be formulated as $[o_1 \not\prec o] \wedge [o_2 \not\prec o] \wedge \cdots$. In particular, the conjunct $o_1 \not\prec o$ is denoted as the disjunction of at most d expressions, e.g., $[o.[1] > o_1.[1]] \vee \cdots \vee [o.[d] > o_1.[d]]$. Here, it is assumed the larger the attribute value, the better. Accordingly, the condition in this paper is represented by the conjunctive normal form (CNF).

Naturally, there appears another question along with the analysis: i.e., how do we derive $\mathcal{D}(o)$ efficiently? Upon the dominator set $\mathcal{D}(o)$ is obtained, the condition of o can be easily derived in CNF form. Thus, we define the dominator set $\mathcal{D}(o)$ in Definition 5 in order to further decompose the mission of the dominator set derivation.

Definition 5. **(The dominator set)**. Given a skyline query over an incomplete dataset $\mathcal{O}$. For each object $o \in \mathcal{O}$, the dominator set $\mathcal{D}(o)$ (consisting of all objects that possibly dominate o) can be derived by Eq. 1, where d is the number of data attributes and O_i represents the set of objects whose i-th dimensional values are missing.

$$\mathcal{D}(o) = \bigcap_{i=1}^{d} D_i(o) \tag{1}$$

$$D_i(o) = \begin{cases} \{p \in \mathcal{O} - \{o\} \mid o.[i] \leq p.[i]\} \cup O_i & \text{if } o.[i] \text{ is observed} \\ \mathcal{O} - \{o\} & \text{otherwise} \end{cases} \tag{2}$$

The set $D_i(o)$ is composed of objects whose i-th attribute values are missing or not worse than that of o, if o has an observed value in attribute i. Otherwise, if o misses its value in the attribute i, $D_i(o)$ is set as the super set $(\mathcal{O}-\{o\})$. Thus, the intersection set of $D_i(o)$s $(i = 1, \cdots, d)$ contains objects that are missing or not worse than o in every attribute, which definitely includes all objects that have possibilities to dominate o, i.e., forming the dominator set $\mathcal{D}(o)$. For ease of understanding, Table 4 lists the dominator set for every object over our sample dataset (depicted in Table 1).

TABLE 4
The Dominator Sets for Objects on the Sample Dataset

Object	o_1	o_2	o_3	o_4	o_5
$\mathcal{D}$	$\{o_5\}$	$\varnothing$	$\varnothing$	$\{o_2, o_5\}$	$\{o_1, o_2\}$

4.2 Getting the C-Table

On top of the derived dominator set $\mathcal{D}(o)$, Algorithm 2 gives the procedure of constructing c-table for a skyline query. It takes as inputs an incomplete dataset $\mathcal{O}$, a query Q, and a pruning threshold α, and outputs the c-table, denoted as $\mathcal{C}$.

To begin with, it initializes $\mathcal{C}$ as an empty set (line 1). Then, for each object $o \in \mathcal{O}$, it derives its dominator set $\mathcal{D}(o)$ based on Definition 5 (line 3). If there is no object in $\mathcal{D}(o)$, it indicates that the object o is certainly a skyline object. Hence, in this case, the condition $\phi(o)$ is assigned as the value of *true* (lines 4-5). On the contrary, if there are a large fraction of objects in the dominator set $\mathcal{D}(o)$, signifying a lot of objects are possible to dominate o. As a result, we use a threshold α to identify the case that o is very likely to be dominated. If $|\mathcal{D}(o)| > \alpha \cdot |\mathcal{O}|$, we deem o is not a skyline object, and set $\phi(o)$ as the value of *false* (lines 6-7). In addition, if there exists an object $o' \in \mathcal{D}(o)$ dominating o according to Definition 1 (where both o and o' have no missing attribute value), then o is not a skyline object. Thus, the condition $\phi(o)$ is set as the value of *false* (lines 8-9). Otherwise, we generate the condition $\phi(o)$ in the form of $[o_1 \not\prec o] \wedge [o_2 \not\prec o] \wedge \cdots$, with each conjunct $o_i \not\prec o$ as $[o.[1] > o_i.[1]] \vee \cdots \vee [o.[d] > o_i.[d]]$, where $o_i \in \mathcal{D}(o)$ for $i = 1, 2, \cdots$ (line 11). Whatever the condition $\phi(o)$ is, it is added to the c-table $\mathcal{C}$ (line 12). Finally, the algorithm stops after returning the c-table $\mathcal{C}$ (line 13).

We would like to highlight that, the usage of parameter α is necessary to prune the case that o is more likely to be dominated. When $\mathcal{D}(o)$ is large, (i.e., o is probably dominated by too many objects), the probability of o being a query answer object will be low, even near to zero. Moreover, a large $\mathcal{D}(o)$ results in a complex condition $\phi(o)$ (that contains too many conjuncts), which hinders significantly the probability computation of $\phi(o)$. In this case, even though getting the probability of the complex condition, it brings limited

4. Banjo is available at https://users.cs.duke.edu/ amink/software/banjo/.
5. Infer.Net is available at http://infernet.azurewebsites.net/.

1041-4347 (c) 2019 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TKDE.2019.2946798, IEEE Transactions on Knowledge and Data Engineering

Algorithm 2: Get-CTable

Input: an incomplete dataset $\mathcal{O}$, a query Q, a pruning threshold α
Output: the c-table $\mathcal{C}$

$\mathcal{C} \longleftarrow \varnothing$
foreach object $o \in \mathcal{O}$ **do**
  derive its dominator set $\mathcal{D}(o)$ using Eq. 1
  if $|\mathcal{D}(o)| = 0$ **then**
    $\phi(o) \longleftarrow$ *true* `// o is a skyline object`
  else if $|\mathcal{D}(o)| > \alpha \cdot |\mathcal{O}|$ **then**
    $\phi(o) \longleftarrow$ *false* `// o is deemed not to be a skyline object`
  else if $\exists o' \in \mathcal{D}(o)$ dominates o **then**
    $\phi(o) \longleftarrow$ *false* `// o is not a skyline object`
  else
    generate the condition $\phi(o)$ of o
  $\mathcal{C} \longleftarrow \mathcal{C} + \{\langle o, \phi(o)\rangle\}$
return $\mathcal{C}$

Algorithm 3: Adaptive DPLL Search (ADPLL)

Input: a condition ϕ, the probability *prob*
Output: the probability of the condition $\Pr(\phi)$

$\Pr(\phi) \longleftarrow 0$
if the conjuncts in ϕ are independent || ϕ = *true* or *false* **then**
  compute the probability of ϕ, i.e., $\Pr(\phi)$
  return *prob*$\cdot \Pr(\phi)$
select a variable v that appears the most times in ϕ
foreach assignment v_a of v **do**
  $\phi' \longleftarrow \phi(v = v_a)$
  $\Pr(\phi) \longleftarrow \Pr(\phi) +$ ADPLL(ϕ', *prob*$\cdot p(v_a)$)
return $\Pr(\phi)$

improvement on query accuracy. In other words, what we gain cannot compensate for what we pain. Hence, it is unnecessary to spend much cost in deriving the probability of those complex conditions.

Furthermore, the smaller the threshold value of α, the more the objects pruned and the more efficient the algorithm. Informally, in the ideal situation, the parameter value of α should approach, from a very small number closely to zero, the ratio of the number of non-answers to the data set cardinality. For the case of the *large dominator set* and/or *high missing rate*, it fits a large α value. Otherwise, it is prone to filter out more objects that are probably true answers, and thereby results in a lower query accuracy. As observed from our experimental evaluation, a relatively smaller value of α (e.g., 0.01) suffices to support the crowd query. A big value of α degrades remarkably query efficiency, yet very slightly improving query accuracy.

***Example* 2.** For the skyline query over the sample dataset, we have shown the c-table and the dominator sets in Table 3 and Table 4, respectively. Now, we are going to explain how to build the c-table (i.e., getting those conditions) based on the dominator sets. First, for the object o_1 (from the sample dataset depicted in Table 1), we could get $\mathcal{D}(o_1)$ (being $\{o_5\}$, as shown in Table 4). It means that only the object o_5 has the possibility to dominate o_1, according to the dominance relationship definition on complete data (as stated in Definition 1). In other words, if o_1 defeats o_5 in at least one attribute, then o_5 would not dominate o_1, and thus, o_1 would be a skyline object. Thus, the condition of o_1, i.e., $\phi(o_1)$, is written as $(Var(o_5,\ a_2) < 2) \vee (Var(o_5,\ a_3) < 3) \vee (Var(o_5, a_4) < 4)$, as depicted in Table 3. Then, for the object o_2, we set the condition of o_2, i.e., $\phi(o_2)$, as true, due to the empty $\mathcal{D}(o_2)$ set. It indicates that there is no object dominating o_2. According to Definition 2, the object o_2 is a skyline object. Similarly, we can derive that, the condition $\phi(o_3)$ gets the value of true. As shown in Table 3, the condition $\phi(o_4)$ is $(Var(o_2,\ a_2) < 3) \wedge [(Var(o_5, a_2) < 3) \vee (Var(o_5, a_3) < 1) \vee (Var(o_5, a_4) < 2)]$. The condition $\phi(o_5)$ is $[(Var(o_5, a_2) > 2) \vee (Var(o_5, a_3) > 3) \vee (Var(o_5, a_4) > 4)] \wedge [(Var(o_5, a_2) > Var(o_2, a_2)) \vee (Var(o_5, a_3) > 2) \vee (Var(o_5, a_4) > 2)]$.

Let $|\mathcal{O}|$ be the dataset cardinality, $|\mathcal{D}|$ be the average number of objects in dominator sets, and d be the number of attributes in the dataset. The complexity of getting dominator sets is $O(d \cdot |\mathcal{O}|^2)$. In addition, as every condition has $|\mathcal{D}|$ conjuncts with each conjunct having at most d disjuncts (i.e., expressions), it needs $O(d \cdot |\mathcal{D}| \cdot |\mathcal{O}|)$ to generate the conditions. Thus, the overall complexity of c-table construction is $O(d \cdot |\mathcal{O}|^2)$.

5 Probability Computation

In this section, we present an efficient algorithm, called ADPLL, for probability computation.

An intuitive solution (called Naive) to compute $\Pr(\phi(o))$ is to evaluate all the variable value combinations (i.e., assignments) of the variables (w.r.t., missing values) in $\phi(o)$, and to aggregate the probabilities of those assignments with the value of true for getting $\Pr(\phi(o))$. One can easily find that it is a #SAT problem if the variables can only get the values of 0 or 1 randomly. The #SAT problem is also known as the (weighted) model counting problem [31], which is a #P-complete problem. In BayesCrowd, the continuous or infinite attribute values have been conducted the discretization at the preprocessing step. Hence, in our probability computation problem, the variables can get a group of discrete values under certain distributions (not just the values of 0 or 1). It indicates that the problem is at least as hard as the #SAT problem.

To this end, we resort to the solvers of the #SAT problem, in order to address our problem efficiently. DPLL (abbreviation of Davis-Putnam-Logemann-Loveland) search [32], [33] is a popular and efficient solver for the #SAT problem, which is able to derive the accurate probability of the formula for the case of the variables only randomly being the values of 0 or 1. As a result, we propose an *adaptive* DPLL (ADPLL for short) algorithm for probability computation. ADPLL recursively selects the most frequent variable in a condition ϕ, and breaks the conjunct correlation in the condition as quickly as possible. The probability of the independent conjuncts can be directly calculated, and thereby boosting computation efficiency.

Algorithm 3 shows the pseudo-code of ADPLL algorithm. For a condition ϕ of an object with an initial prob-

1041-4347 (c) 2019 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TKDE.2019.2946798, IEEE Transactions on Knowledge and Data Engineering

ability *prob*, ADPLL algorithm is responsible for obtaining the probability of the condition, i.e., $\Pr(\phi)$. Note that, the parameter *prob* is initialized as one when ADPLL algorithm is invoked for the first time. Specifically, ADPLL initializes the probability $\Pr(\phi)$ to zero (line 1). If ϕ is *true* (or *false*), ADPLL returns $\Pr(\phi)$ as 1 (or 0). If the conjuncts in ϕ are independent (i.e., each conjunct in ϕ has different variables), ADPLL directly computes and returns $\Pr(\phi)$ via leveraging the special conjunctive rule and the general disjunctive rule (lines 2-4). To be more specific, the special conjunctive rule claims that, for two conjuncts p and q, if they are independent, the probability $\Pr(p \wedge q) = \Pr(p) \cdot \Pr(q)$. In addition, the general disjunctive rule is, for any two disjuncts p and q, the probability $\Pr(p \vee q) = 1 - \Pr(\neg p \wedge \neg q)$, which is employed to derive the probability of each conjunct in the condition. When conjuncts in the condition ϕ are correlated, ADPLL selects a variable v if it occurs the most times in ϕ. A random selection breaks the tie if the occurrence times of variables are identical (line 5).

In the sequel, for each possible value v_a of the variable v, ADPLL first gets a new condition ϕ' by substituting v_a for the variable v in ϕ (line 7). Then, it updates $\Pr(\phi)$ by adding the probability value returned by ADPLL(ϕ', $prob \cdot p(v_a)$) (line 8). Here, $p(v_a)$ denotes the probability of getting the value v_a for v under the learned data distribution in the preprocessing step. At that time, ADPLL is recursively invoked, by taking as inputs the new updated ϕ' and the updated probability $prob \cdot p(v_a)$. It is worth noting that, when the probability $\Pr(\phi)$ can be calculated directly (i.e., ϕ = *true* or *false*, or conjuncts in ϕ are independent), ADPLL returns $\Pr(\phi)$ at line 4 of Algorithm 3. Then, it traces back to the previous layers of ADPLL recall (if exists), and continues processing the "foreach" loop of Algorithm 3.

Example 3. Take the condition of o_5, i.e., $\phi(o_5)$ shown in Table 3, as an example. For simplicity, assume that the probability distribution of each attribute is as follows. The probability of attribute a_2 getting the value i, i.e., $p(a_2 = i)$, is 0.1 for $i = 0, 1, \cdots, 9$. Similarly, the probability $p(a_3 = i)$ is equal to 0.125 for $i = 0, 1, \cdots, 7$. The probability $p(a_4 = i)$ equals 0.1 for $i = 0, 1, 5$, is 0.2 for $i = 2, 3$, and is 0.3 for $i = 4$. The parameter $prob$ in ADPLL is initialized to one. First of all, since each of the three variables $Var(o_5, a_2)$, $Var(o_5, a_3)$, and $Var(o_5, a_4)$ occurs twice in $\phi(o_5)$ (as depicted in Table 3), ADPLL randomly picks one of them, e.g., the variable $Var(o_5, a_4)$. Then, ADPLL assigns one possible value to it, e.g., $Var(o_5, a_4)$ = 0. Note that, according to the probability distribution of the attribute a_4, the probability of getting the value 0, i.e., $p(Var(o_5, a_4) = 0)$, is 0.1. As a result, when the value of 0 substitutes for $Var(o_5, a_4)$ in the condition $\phi(o_5)$, $\phi'(o_5)$ becomes $[(Var(o_5, a_2) > 2) \vee (Var(o_5, a_3) > 3)] \wedge [(Var(o_5, a_2) > \text{Var}(o_2, a_2)) \vee (Var(o_5, a_3) > 2)]$. Taking the condition $\phi'(o_5)$ and $prob \cdot p(Var(o_5, a_4) = 0)$ (being 0.1) as inputs, another ADPLL search is invoked.

In the second layer of ADPLL search, assume that $Var(o_5, a_3)$ is selected and assigned the value of 0. Based on the supposed probability distribution of a_3, the probability $p(Var(o_5, a_3) = 0)$ equals 0.125. At that time, $\phi''(o_5)$ becomes $(Var(o_5, a_2) > 2) \wedge (Var(o_5, a_2) > \text{Var}(o_2, a_2))$, after the value of 0 substitutes for $Var(o_5, a_3)$ in the condition ϕ'. Another new ADPLL search is recalled with parameters $\phi''(o_5)$ and the probability (0.1×0.125). When entering the third layer of ADPLL search, assume that $Var(o_5, a_2)$ is chosen and assigned the value of 0, where $p(Var(o_5, a_2) = 0)$ is equal to 0.1. Afterwards $\phi'''(o_5)$ turns false. One new ADPLL search is recalled with parameters $\phi'''(o_5)$ and the probability $(0.1 \times 0.125 \times 0.1)$. During the forth layer of ADPLL search, it gets $\Pr(\phi'''(o_5))$ as 0. Then, it goes back to previous layers of ADPLL search step by step. It finally returns the probability $\Pr(\phi(o_5))$ as 0.823.

Let d be the number of attributes, $|\mathcal{D}|$ be the average number of objects in the dominator sets, and N is the average number of the possible values that one variable could get. As mentioned earlier, the problem of probability computation is at least as hard as #SAT problem. The Naive method is of complexity $O(N^{d \cdot |\mathcal{D}|})$ for computing the probability of each condition. Instead, ADPLL in the recursive style makes the expression correlation weaker and weaker, and it is guaranteed to obtain the accurate probability based on the aforementioned two rules. Although ADPLL will degrade to Naive at the worst case, ADPLL is empirically confirmed to be much faster than Naive. It means that, the worst case occurs at a very small probability in ADPLL.

We have also generalized the approximate weighted ApproxCount algorithm [34] for the probability computation problem. It turns out that the approximate solution performs worse than ADPLL in terms of both efficiency and accuracy. Toward this, we conclude two major reasons as follows. (i) Due to the ability of weakening expression correlation in the condition, the probability of the condition is easily calculated by ADPLL. (ii) In the approximate solution, in order to sample an assignment that makes the condition get the value of *true*, the multiple-value variables (in probability computation) lead to much more overhead.

6 Crowd Task Selection

In this section, we introduce three task selection strategies, and discuss the complexity of BayesCrowd.

6.1 Budget Constraint and Latency Concern

Paying crowd workers to complete all tasks is prohibitively expensive. Thus, in many real-life scenarios, it is most likely to spend an amount of money not larger than the budget constraint (denoted by B) on the crowdsourcing work. For a group of similar tasks (with comparable difficulties), crowdsourcing each of those tasks is assumed to spend a fixed amount of money. Hence, in this paper, the budget B refers to the number of tasks that a requester could afford. In contrast, when it comes to the case of variable task difficulties, one could accumulate the respective crowd cost of the task one by one for budget concern, which impacts the proposed solution negligibly.

On the other hand, the latency is also a key factor that the requester usually concerns. For example, the requester would like to finish the task within limited time units. In this paper, we borrow the idea from [35], [36] that, the latency is measured by the number of iterations in the crowd

1041-4347 (c) 2019 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TKDE.2019.2946798, IEEE Transactions on Knowledge and Data Engineering

query. The tasks posted in one iteration are thought to be completed almost at the same time. Therefore, adjusting the number of tasks in one iteration could control the latency of the crowd query if the total number of crowd tasks is fixed.

As a result, we choose to post crowd tasks *in batches*. This batch/iteration policy provides an opportunity to take into account the new-round returned task answers for the following task selection timely in BayesCrowd. In addition, since the *conflicted* tasks may lead to unreasonable results, the crowd tasks in one iteration must avoid conflictions. In our implementation, we handle conflictions by identifying the variables in chosen tasks (i.e., expressions), and make sure that any pair of chosen tasks in one iteration does not share the same variable.

6.2 Task Selection Strategies

Under the *iteration* policy, we select crowd tasks in each iteration through two steps, including (i) choosing top-k objects with high possibilities to be query answers, and (ii) choosing an expression among the condition from each of chosen top-k objects in previous step. Here, the parameter k is decided by the number of awaiting crowd tasks per round under budget and latency constraints. In what follows, we details the two steps.

Specifically, among each iteration, at the first step, we employ Shannon entropy as a metric to quantify the uncertainty of objects being the query result objects. Specifically, the entropy of an object in terms of its confidence $\Pr(\phi(o))$ (denoted as p_o for short) is defined by Eq. 3.

$$H(o) = -(p_o \cdot \log_2 p_o + (1 - p_o) \cdot \log_2(1 - p_o)) \quad (3)$$

Intuitively, if the probability $\Pr(\phi(o))$ is skewed towards 0 or 1, we expect to predict whether the object o is a query result object with reasonable accuracy. On the other hand, if $\Pr(\phi(o))$ is a fair coin flip, we have no reliable information about whether o is a query result object. Thus, we choose the top-k objects with the highest entropy values.

In the second step, we introduce several effective strategies to select expressions, i.e., crowd tasks.

Frequency based strategy (FBS). For each of the top-k objects o chosen in the first step, we sort expressions in the condition $\phi(o)$ in the non-ascending order of expression appearance times in conditions of the chosen top-k objects (i.e., the expression frequency). Then, we select the most frequent expression as the crowd task w.r.t. o.

Let d be the number of attributes, and $|\mathcal{D}|$ be the average number of objects in dominator sets. Since there are at most $d \cdot |\mathcal{D}|$ expressions in one condition, it takes $O(k \cdot d \cdot |\mathcal{D}|)$ time for frequency derivation. If n denotes the number of different expressions in those $k \cdot d \cdot |\mathcal{D}|$ expressions, the complexity of ranking those expressions is $O(n \log_2 n)$. As a result, FBS is of the complexity $O(k \cdot d \cdot |\mathcal{D}| + n \log_2 n)$ per iteration.

Utility based strategy (UBS). We introduce *a marginal utility function* to measure the benefit of crowdsourcing one task. It is defined as the *information gain* based on the entropy function, as stated in Definition 6.

Definition 6. **(The marginal utility function)**. Given an incomplete dataset $\mathcal{O}$ and a query Q. For a condition $\phi(o)$ of the object $o \in \mathcal{O}$, the (expected) marginal utility function, denoted as $\mathcal{G}(o, e)$, of selecting an expression e (i.e., task) from $\phi(o)$ to crowdsource is defined in Eq. 4.

$$\mathcal{G}(o, e) = H(o) - \mathbb{E}[H(o|e)] \quad (4)$$

$$\mathbb{E}[H(o|e)] = \Pr(e) \cdot H(o|e = \text{true}) + (1 - \Pr(e)) \cdot H(o|e = \text{false}) \quad (5)$$

Here, $\Pr(e)$ represents the probability of the expression e being satisfied, and $H(o|e = \text{true/false})$ denotes the entropy of the object o after the expression e in the condition $\phi(o)$ gets the value of *true/false*.

Indeed, $\mathcal{G}(o, e)$ measures the expected quality improvement for the object o when crowdsourcing the expression e in the condition $\phi(o)$. When an expression is determined, the corresponding condition can be simplified.

Hence, different from FBS, for each of the chosen top-k objects o, UBS selects the expression in $\phi(o)$ with the highest marginal utility at the second step. It is not hard to predict that, UBS is very likely to obtain higher query accuracy under the help of the marginal utility function, while it is less efficient due to multiple probability computation (resulting from the marginal utility derivation). Let N be the number of values that one variable can have. UBS needs $O(k \cdot d \cdot |\mathcal{D}| \cdot N^{d \cdot |\mathcal{D}|})$ time per iteration at the worst case to select k crowd tasks.

The hybrid heuristic strategy (HHS). HHS combines the advantages of FBS and UBS. Like FBS, for the condition

Algorithm 4: Hybrid Heuristic Strategy (HHS)

Input: an incomplete dataset $\mathcal{O}$, a query Q, a budget B, a latency constraint L, a parameter m
Output: the query result set $\mathcal{R}$

$\mathcal{R} \longleftarrow \varnothing$; $\mu \longleftarrow \lceil \frac{B}{L} \rceil$
while $B \neq 0$ and there exists an expression in conditions **do**
  $\mathcal{T} \longleftarrow \mathcal{O}_t \longleftarrow \varnothing$
  foreach object $o \in \mathcal{O}$ **do**
    $p_o \longleftarrow$ ADPLL($\phi(o)$, 1) `// Algorithm 3`
    $H(o) \longleftarrow -p_o \cdot \log_2 p_o - (1 - p_o) \cdot \log_2(1 - p_o)$ `// Eq. 3`
  add $\min(B, \mu)$ objects o with highest $H(o)$ into $\mathcal{O}_t$
  $B \longleftarrow \max(B - \mu, 0)$
  foreach object $o \in \mathcal{O}_t$ **do**
    $g \longleftarrow c \longleftarrow 0$
    sort expressions in $\phi(o)$ in the order of frequencies
    foreach expression $e \in \phi(o)$ **do**
      $\mathbb{E}[H(o|e)] \longleftarrow \Pr(e) \cdot H(o|e =$ true$) + (1 - \Pr(e)) \cdot H(o|e =$ false$)$ `// Eq. 5`
      $\mathcal{G}(o, e) \longleftarrow H(o) - \mathbb{E}[H(o|e)]$ `// Eq. 4`
      if $\mathcal{G}(o, e) > g$ **then**
        $g \longleftarrow \mathcal{G}(o, e)$; $e^\star \longleftarrow e$
        $c \longleftarrow 0$
      else
        $c \longleftarrow c + 1$
        if $c = m$ **then**
          break
    $\mathcal{T} \longleftarrow \mathcal{T} + \{e^\star\}$
  post tasks in the set $\mathcal{T}$ on the crowdsourcing market
  get task answers from crowd workers
  update conditions in c-table $\mathcal{C}$ using new answers
  infer the query answer set $\mathcal{R}$
return $\mathcal{R}$

1041-4347 (c) 2019 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TKDE.2019.2946798, IEEE Transactions on Knowledge and Data Engineering

$\phi(o)$ of each chosen object o in the first step, HHS visits the expressions in the order of their frequencies. Like UBS, HHS calculates the expected marginal utility of selecting an expression. What's more attractive is that, we introduce a parameter m for HHS to enable a heuristic, which helps to control the computation cost. Specifically, as long as HHS does not get a larger utility score when evaluating consecutive m expressions from a condition $\phi(o)$, the current expression with the biggest score is directly selected as the crowd task w.r.t. o, and HHS stops evaluating the remaining expressions in $\phi(o)$.

Algorithm 4 depicts the pseudo-code of the crowdsourcing phase of BayesCrowd using HHS. It takes as inputs an incomplete dataset $\mathcal{O}$, a query Q, a budget B, a latency constraint L, and the parameter m of HHS. The set of query answers, denoted by $\mathcal{R}$, is returned. HHS first empties the result set $\mathcal{R}$, and estimates the number of tasks per batch/iteration, denoted as μ, via computing $\lceil \frac{B}{L} \rceil$ (line 1). Then, it starts to choose crowd tasks iteratively (lines 2-26). Specifically, it invokes the function ADPLL to select $\min(B, \mu)$ objects with highest entropies. The selected objects are collected into the set $\mathcal{O}_t$. Meanwhile, the budget B is updated (lines 3-8). Thereafter, for each object $o \in \mathcal{O}_t$, HHS visits the expression e from $\phi(o)$ in the non-ascending order of expression frequencies (lines 9-22). The temporal variable c is used to facilitate the condition verification of enabling the heuristic. HHS attempts to get the expression $e^\star$ from $\phi(o)$ that has the largest expected utility improvement, denoted by g. As a heuristic, if the consecutive m expressions have no larger expected utility improvement than g, HHS directly adds the *current* expression $e^\star$ to the set $\mathcal{T}$, which is employed to collect the awaiting crowd tasks in this iteration (lines 15-22).

In the sequel, tasks in $\mathcal{T}$ are posted on the crowdsourcing market. When the answers are returned from crowd workers, the algorithm updates conditions of c-table $\mathcal{C}$ using new obtained answers (lines 23-25). Upon the update, some conditions will turn true or false, some shall be simplified or remain the same. It also infers the result set $\mathcal{R}$ (line 26). This process proceeds until the budget B is used up or there is no expression in conditions. Finally, it returns the query result set $\mathcal{R}$ and terminates (line 27).

Example 4. To illustrate the task selection using HHS, we take the skyline query on the sample dataset as an example. For simplicity, it is assumed that the budget B is 6 and the latency constraint L is 3, indicating that two tasks should be posted to crowd workers per iteration. The parameter of m is set as 2.

First, regarding the condition of each object shown in Table 3, HHS computes the entropy of each object. The conditions of o_2 and o_3 have gotten the true value (i.e., the entropy is zero). Currently, the result set $\mathcal{R}$ is $\{o_2, o_3\}$. The entropy of o_1, i.e., $H(o_1)$, is 0.72. The entropy $H(o_4)$ is 0.62, and the entropy $H(o_5)$ is 0.67. Hence, HHS picks objects o_1 and o_5 (with highest entropies) to form the set $\mathcal{O}_t$. Then, for each of the two objects, HHS is going to select one task from its condition. Specifically, for the object o_1, there are three expressions (denoted by e_1, e_2, and e_3) in the condition $\phi(o_1)$, namely, e_1 is $\mathrm{Var}(o_5, a_2) < 2$, e_2 is $\mathrm{Var}(o_5, a_3) < 3$, and e_3 is $\mathrm{Var}(o_5, a_4) < 4$. Obviously, for each of e_1, e_2, and e_3, its frequency is one. Thus, a random ranking is used to break the tie, e.g., e_1, e_2, and e_3 in order.

TABLE 5
The Updated C-Table

Object	Condition
o_1	true
o_2	true
o_3	true
o_4	$(\mathrm{Var}(o_2, a_2) < 3) \wedge [(\mathrm{Var}(o_5, a_2) < 3) \vee (\mathrm{Var}(o_5, a_4) < 2)]$
o_5	$\mathrm{Var}(o_5, a_2) > 2$

In the following, the marginal utility of each expression is derived, i.e., $\mathcal{G}(o_1, e_1)$ is 0.072, $\mathcal{G}(o_1, e_2)$ is 0.157, and $\mathcal{G}(o_1, e_3)$ is 0.322. Hence, the expression e_3 is chosen to crowdsource. Obviously, there is no opportunity for the heuristic in HHS (with the parameter of m) to exhibit its power. In the similar way, for the condition $\phi(o_5)$, the expression $\mathrm{Var}(o_5, a_3) > 3$ is selected. In the sequel, the two tasks are posted on the crowdsourcing market. Assume that the returned answers are $\mathrm{Var}(o_5, a_4) < 4$ and $\mathrm{Var}(o_5, a_3) = 3$. Using those answers, HHS updates the initial c-table (shown in Table 3) as a new c-table (depicted in Table 5). The result set $\mathcal{R}$ is updated as $\{o_1, o_2, o_3\}$. Thereafter, HHS enters the second iteration, and sets $\mathcal{O}_t$ as $\{o_5, o_4\}$ due to $H(o_4)$ being 0.63 and $H(o_5)$ being 0.88. Similarly, it chooses the expressions $\mathrm{Var}(o_5, a_2) > 2$ and $\mathrm{Var}(o_2, a_2) < 3$ from conditions of o_5 and o_4, respectively. Assume that it is confirmed by crowd workers that, $\mathrm{Var}(o_5, a_2) > 2$ and $\mathrm{Var}(o_2, a_2) > 3$. Finally, $\phi(o_4)$ becomes false, and $\phi(o_5)$ turns true.

7 Experimental Evaluation

In this section, we present a comprehensive experimental evaluation. In what follows, we first describe experimental settings, and then, we report experimental results and our findings. All algorithms were implemented in Java language, and all experiments were conducted on an Intel Core i7 Duo 3.60GHz PC with 28GB RAM, running Microsoft Windows 7 Professional Edition.

In our experiments, we utilize two datasets including *NBA* and *Synthetic*. *NBA*[6] is a real dataset containing 10,000 records of player competition information from 1979 to 2004. We use eleven attributes including total points, total rebounds, etc. In addition, for scalability evaluation, we generate the *Synthetic* dataset with 100,000 records and nine attributes. It shares the same Bayesian network with the typical Adult[7] dataset from UCI Machine Learning Repository. As mentioned previously, the frameworks Banjo and Infer.Net are employed to train Bayesian networks. Following the previous studies on incomplete data [5], [6], we delete attribute values randomly to simulate incomplete datasets, unless otherwise indicated.

We verify the performance of the following algorithms in our experiments. (i) Get-CTable algorithm (i.e., Algorithm 2) for c-table construction. (ii) Baseline algorithm, which derives the dominator sets through simple pairwise comparisons between objects for c-table construction. (iii)

6. http://www.nba.com.
7. https://archive.ics.uci.edu/ml/datasets/adult.

1041-4347 (c) 2019 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TKDE.2019.2946798, IEEE Transactions on Knowledge and Data Engineering

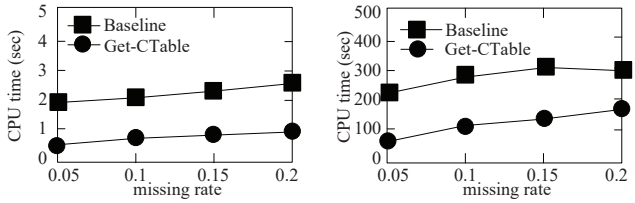


Fig. 2. Evaluation on c-table construction

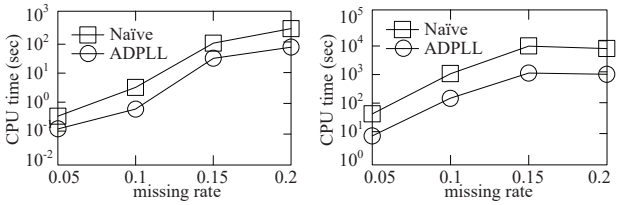
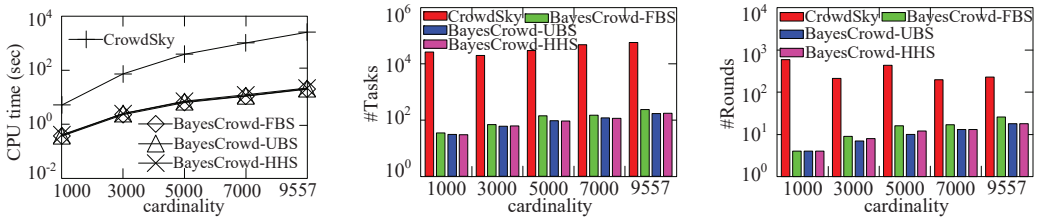


Fig. 3. Evaluation on probability computation



(a) Time cost (b) Monetary cost (c) Latency

Fig. 4. Performance comparison with CrowdSky

ADPLL algorithm (i.e., Algorithm 3) for probability computation. (iv) Naive method, which computes the probability in a brute-force way, as described in Section 5. (v) BayesCrowd-FBS, which adopts BayesCrowd framework, and selects tasks using FBS. (vi) BayesCrowd-UBS, which adopts BayesCrowd framework, and selects tasks via UBS. (vii) BayesCrowd-HHS, which adopts BayesCrowd framework, and selects tasks using HHS (i.e., Algorithm 4). (viii) CrowdSky [23], which is the state-of-the-art crowd skyline method that utilizes the dominating set to prune some tasks, and employs a partitioning approach and skyline layers to support parallelization. It is noteworthy that, since the dominance relationship on *complete* data is utilized in our work, existing skyline algorithms on incomplete data cannot directly support skyline computation with crowdsourcing.

We study the effect of various factors on algorithm performance, including the dataset missing rate, the threshold value of α, the amount of budget (i.e., the number of affordable tasks), the parameter m, the latency (i.e., the number of task selection rounds), and the dataset cardinality. In particular, the missing rate of the data set is defined as the ratio of the total number of missing attribute values to the overall number of (both observed and missing) attributes, which is set as 0.1 by default. The missing rate of each object is roughly equal to the missing rate of the dataset, as the missing information is injected into the dataset randomly over objects and attributes in the experiments. In addition, due to the large difference between the two dateset scales, we set α, the budget, m, and the latency to 0.003, 50, 15, and 5 on *NBA* dataset, respectively. On *Synthetic* dataset, we set them as 0.01, 1000, 50, and 10, respectively. For each set of experiments, we change one of the factors, and set the rest as default values.

In the evaluation, we employ F1 score to measure the query accuracy. Specifically, the query result derived based on the corresponding *complete* data is regarded as the ground truth in the experiments. The returned objects with either true conditions or larger than 0.5 probability being query answers form the query result set. In addition, for each set of experiments, we use the *majority voting* strategy to get task answers, and each task is assigned to three workers. In order to exactly and fairly study the impact of a specific factor on the algorithm performance, the worker accuracy is set to 1.0 by default. Otherwise, it might disturb the other factors' evaluation more or less. Of course, we also add an extra set of experiments to explore the impact of worker accuracy on the algorithm performance. In practice, we could select the workers whose accuracies being above one certain value to answer tasks, for controlling the final query answer accuracy (e.g., this kind of worker recruitment is supported by AMT).

7.1 Efficiency of C-Table Construction

The first set of experiments evaluates the efficiency of Get-CTable algorithm, compared with Baseline method. Note that, the typical indices and skyline techniques on complete data are inapplicable due to the data incompleteness.

The experimental results when varying the missing rate on both datasets are depicted in Figure 2. One can observe that, Get-CTable algorithm is obviously faster than Baseline in each case. With the increase of the missing rate, the time cost gradually climbs up. This is because, the dominator set becomes larger with the growth of the missing rate, which incurs more processing cost. Besides, in contrast to the pairwise comparisons in Baseline, Get-CTable algorithm first sorts the values on each dimension, and then derives the dominator set through *fast bitwise* operations, which makes Get-CTable perform much better.

7.2 Efficiency of Probability Computation

In this set of experiments, we investigate the performance of ADPLL algorithm for probability computation, compared with Naive method. We report the total time of probability computation for conditions in the initial c-table.

The experimental results are plotted in Figure 3. We can observe that, ADPLL constantly performs faster than Naive under various missing rates in all cases. This is because, compared with Naive, ADPLL recursively selects the most frequent variable in a condition, and attempts to break

1041-4347 (c) 2019 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TKDE.2019.2946798, IEEE Transactions on Knowledge and Data Engineering

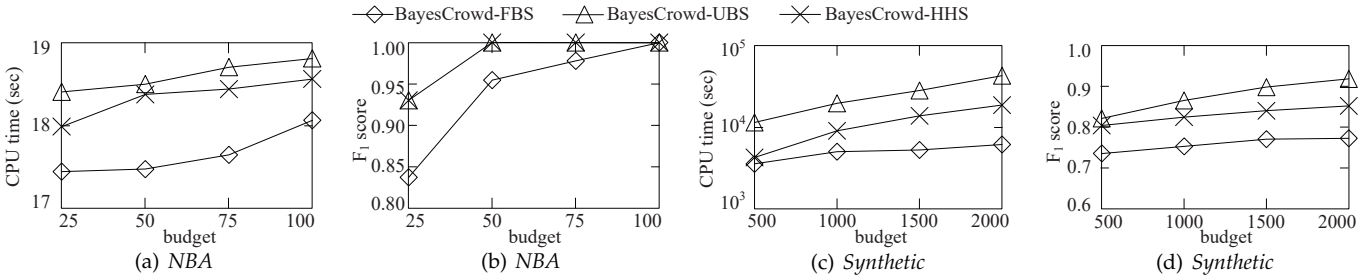


Fig. 5. BayesCrowd cost vs. budget

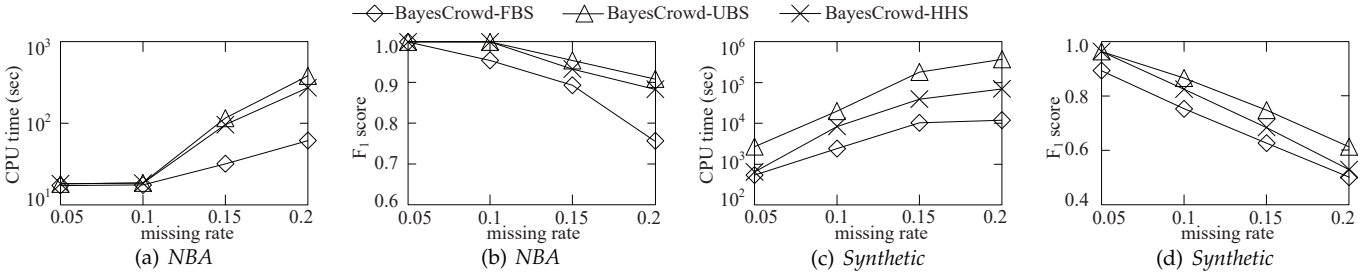


Fig. 6. BayesCrowd cost vs. missing rate

the expression correlation in the condition as quickly as possible, and therefore speeds up computation. Moreover, the efficiency is slowing down with the growth of missing rate. The reason behind is that, there are in average, more expressions and variables in one condition for a higher missing rate. It results in an exponential increase of overhead, which is consistent with the complexity analysis in Section 5.

7.3 Comparison Study

In this set of experiments, we conduct a complete performance comparison between BayesCrowd and the state-of-the-art method CrowdSky [23]. In order to ensure the comparable settings with CrowdSky, we temporally adjust *NBA* dataset by missing all values in two attributes and keeping complete on the other attributes. In addition, since CrowdSky does not support budget constraint, we set a relatively large budget for BayesCrowd (i.e., without budget constraint), and mainly compare the monetary cost (i.e., the number of posted tasks) and the latency (i.e., the number of task selection rounds) of them, where the number of tasks per round is set to 20 for each of them.

When changing the cardinality of *NBA* dataset, the experimental results are shown in Figure 4. First, regarding the execution time (of algorithms, which excludes the time of workers answering tasks), BayesCrowd (using FBS, UBS, and HHS) is up to two orders of magnitude faster than CrowdSky. Moreover, with the growth of the data scale, the time cost of BayesCrowd grows much slower than that of CrowdSky. It confirms the good scalability of BayesCrowd. This is because, using the Bayesian network and the c-table model, BayesCrowd is able to infer some preference information (i.e., better/worse than, or equal to) in tasks, using returned answers per iteration. In contrast, CrowdSky derives skyline objects by collecting all the missing preferences in crowd attributes without any inference, incurring more overhead especially for larger datasets. In addition, the reason on the increasing trend of the cost is that, with the growth of dataset cardinality, there are more missing values and objects in the dataset, which leads to more overhead on deriving dominator sets and selecting tasks.

Figure 4(b) and Figure 4(c) report the total number of posted tasks and the number of task selection rounds incurred by algorithms, respectively. One can observe that, CrowdSky needs at least one order of magnitude more tasks and rounds than BayesCrowd to achieve the query result. In contrast, BayesCrowd (using FBS, UBS, and HHS) requires remarkably less tasks and rounds. It also justifies the conclusion that CrowdSky needs more time to answer the query. On the other hand, considering from another angle, we can find that, compared with CrowdSky, BayesCrowd spends much less monetary cost (w.r.t. #tasks) and execution time in processing the query, and has an obviously shorter latency (w.r.t. #rounds).

7.4 Results on BayesCrowd

Effect of budget. Figure 5 shows the corresponding experimental results when varying the amount of budget. It is obvious that, the accuracy climbs up gradually with larger budget, while the efficiency drops. The reason is straightforward, since it incurs more time to choose more affordable tasks (w.r.t. larger budget). Meanwhile, when getting more task answers, the query result will become more accurate. In terms of the execution time, BayesCrowd-FBS is the fastest one, followed by BayesCrowd-HHS. BayesCrowd-UBS is the slowest. It is consistent with the complexity analysis in Section 6.2. In terms of the accuracy, BayesCrowd-UBS is the best, while BayesCrowd-FBS is the worst. This is because, BayesCrowd-UBS exactly chooses the best tasks using the marginal utility function per round at the most amount of computation cost. In contrast, BayesCrowd-HHS balances the pros and cons of BayesCrowd-UBS and BayesCrowd-FBS, and thus, it is able to obtain quite good accuracy at less time cost than BayesCrowd-UBS.

Effect of missing rate. With varying the missing rate from 0.05 to 0.2, Figure 6 plots the corresponding experi-

1041-4347 (c) 2019 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TKDE.2019.2946798, IEEE Transactions on Knowledge and Data Engineering

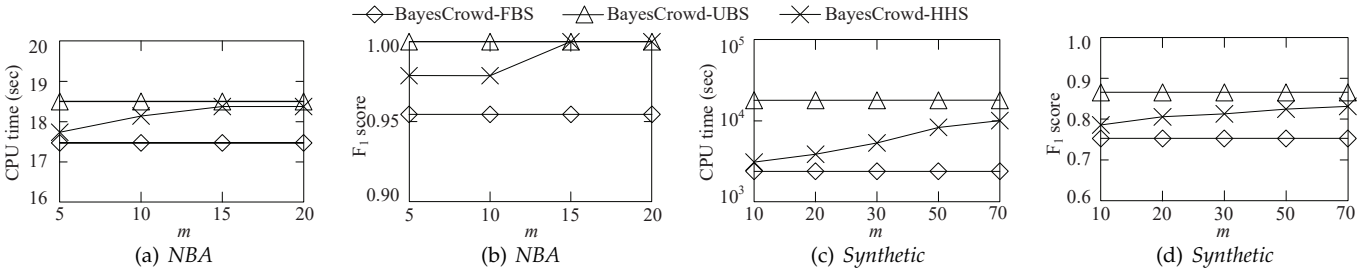


Fig. 7. BayesCrowd cost vs. m

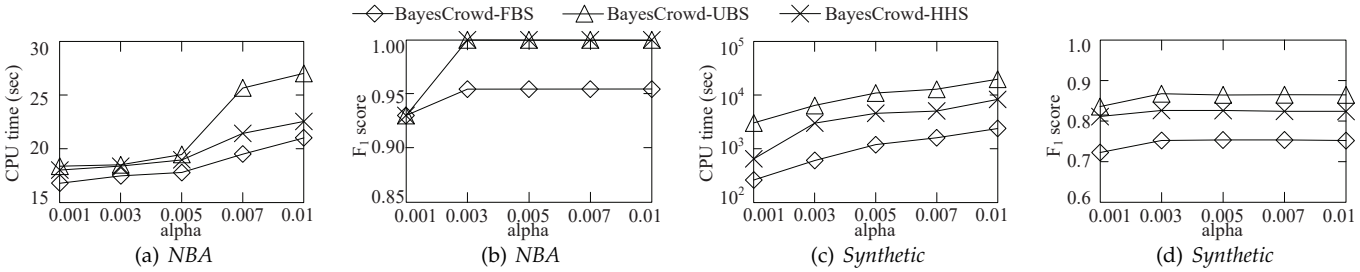


Fig. 8. BayesCrowd cost vs. α

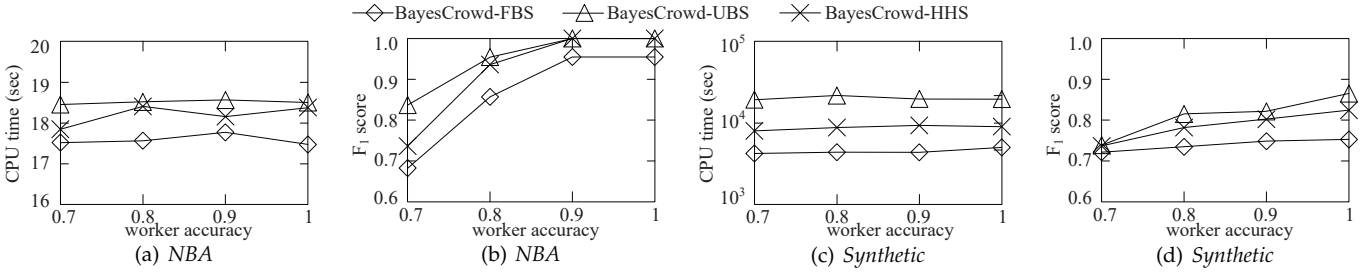


Fig. 9. BayesCrowd cost vs. worker accuracy

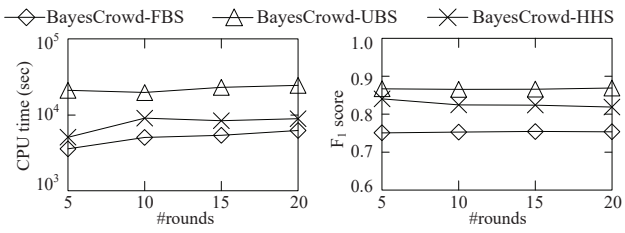


Fig. 10. BayesCrowd cost vs. latency

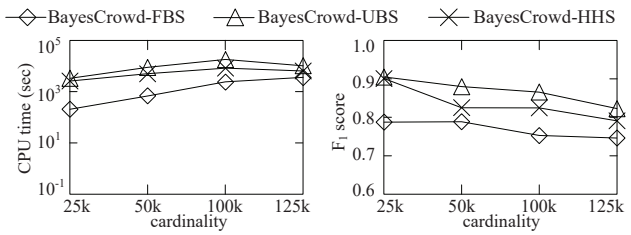


Fig. 11. BayesCrowd cost vs. data cardinality

mental results on two datasets. One can observe that, the time cost increases with the growth of missing rate, and the accuracy is decreasing. The reason behind it is that, there are more expressions (w.r.t. tasks) appearing in the c-table for a higher missing rate, which incurs more overhead. On the other hand, since the budget is fixed, the returned result set contains more uncertainty for the higher missing rate, resulting in lower accuracy. In addition, BayesCrowd-UBS obtains the highest accuracy with the most time consumption. BayesCrowd-HHS has much better accuracy yet more time expenditure than BayesCrowd-FBS in all cases.

Effect of parameter m. Compared with BayesCrowd-FBS/UBS, Figure 7 shows the corresponding experimental results when varying m. It is observed that, for BayesCrowd-HHS, its accuracy gets higher (even near to BayesCrowd-UBS) with the growth of m, while its time cost increases. The reason behind it is that, with the increase of m, BayesCrowd-HHS executes more and more marginal utility computation for selecting each crowd task, and hence, it is more likely for BayesCrowd-HHS to select the most beneficial crowd task, whereas it incurs more overhead simultaneously. In fact, as long as m is large enough, BayesCrowd-HHS is able to get the same performance as BayesCrowd-UBS in both efficiency and accuracy.

Effect of parameter α. When the threshold value α changes from 0.001 to 0.01, we report the corresponding experimental results on both datasets in Figure 8. We observe that, with the growth of the threshold value α, it needs more time to perform BayesCrowd, while the achieved query accuracy gets higher. This is because, for a larger α, the condition of regarding an object being a non-skyline point becomes stricter. In other words, it gets harder to directly prune the object and set its condition as *false*. As a result, there are more complex conditions (with more expressions)

1041-4347 (c) 2019 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TKDE.2019.2946798, IEEE Transactions on Knowledge and Data Engineering

in the c-table, which leads to more overhead for probability computation, but contributes to higher accuracy. Besides, we find that, taking the relatively less time than BayesCrowd-UBS, BayesCrowd-HHS gets pretty good accuracy.

Effect of worker accuracy. Figure 9 depicts the CPU time and the query accuracy when worker accuracy varies from 0.7 to 1.0. Note that, if worker accuracy is 0.8, it means that workers return a correct answer with the confidence 0.8. It is observed that, the time cost is not very sensitive to worker accuracy, while the query accuracy gets higher with the increase of worker accuracy. The reason behind it is easy to analyse. The worker accuracy is only positively related to the query accuracy, and it does not affect the algorithm execution time. It is worth noting that, on the *Synthetic* dataset, BayesCrowd-UBS and BayesCrowd-HHS increase about 10 percent on F1 accuracy with the worker accuracy varying from 0.7 to 1, and the accuracy of BayesCrowd-FBS in Fig. 9(d) actually increases around 3 percent. In contrast, the F1 scores on *NBA* dataset grow more than 20 percent in average for the three strategies. The increasing accuracy volumes on the two datasets are quite different, which mainly attributes to the intrinsic properties of the datasets and the corresponding task difficulties from the two datasets. In addition, similar as previous performance, BayesCrowd-HHS gets near-highest accuracy using less than half time of BayesCrowd-UBS.

Effect of latency. We change the latency (i.e., the number of rounds), and report the experimental results on *Synthetic* in Figure 10. One can find that, the time cost and the accuracy is not very sensitive to latency. This is because, the fixed budget indicates the fixed number of affordable tasks, and hence, the accuracy looks stable, as well as the time cost. Note that, the results on *NBA* are omitted due to the similar performance trends and the space constraint. We can observe that, BayesCrowd is capable of controlling the latency to satisfy the requester's demand.

Effect of data cardinality. We vary the cardinality of *Synthetic*, and present the corresponding experimental results in Figure 11. One can observe that, the time cost goes up with the ascending cardinality. This is because, for more data objects, it needs more time for deriving each dominator set, as well as probability computation for task selection. Thus, it incurs some overhead. Meanwhile, we find that the time cost partly relies on the size of skyline objects. As confirmed, the skyline object size slightly decreases when the cardinality becomes 125K, and thereby alleviating the growth of time cost. In addition, the accuracy decreases gradually with the increase of data cardinality. The reason behind is that, the candidate object set becomes larger, but the budget is fixed. It leads to the descending accuracy.

7.5 Live Experiments on AMT

Last but not the least, we conduct a *practicality study* for BayesCrowd on the live crowdsourcing market AMT. Based on *NBA* dataset with the default parameter settings, the accuracy of BayesCrowd with each of the three task selection strategies is depicted in Table 6. As observed, every strategy has relatively high accuracy, which is comparable to the previous experimental results reported in this section. In addition, it has partly demonstrated that, BayesCrowd is practical in real crowdsourcing marketplaces for skyline queries over incomplete data. Moreover, BayesCrowd has excellent performance especially for high-accuracy workers.

TABLE 6
Live Experiments using NBA dataset on AMT

	BayesCrowd-FBS	BayesCrowd-UBS	BayesCrowd-HHS
F1 score	0.956	0.979	0.978

In summary, ADPLL algorithm benefits probability computation significantly. It makes BayesCrowd practical and efficient. The three task selection strategies FBS, UBS, and HHS have respective features. FBS is the fastest one, and UBS is of the highest accuracy. HHS is the most flexible task selection strategy, i.e., it allows requesters to tune the value of parameter m for balancing the accuracy and algorithm execution time to get satisfying performance. Furthermore, compared with the state-of-the-art method CrowdSky [23], BayesCrowd is demonstrated to perform several orders of magnitude better, no matter in terms of wasted money, algorithm execution time, or latency minimization (with a comparable accuracy).

8 Related Work

Query optimization via crowdsourcing. With the emerging of more and more crowdsourced databases such as CrowdDB [20], Deco [37], and Qurk [38], crowd queries have been extensively explored, e.g., select [39], [40], [41], join [42], [43], [44], [45], sort [42], group-by [46], maximum [47], [48], [49], and top-k query [50], [46], [51], [52], [53], [54]. Also, there are several survey works, e.g., [55], [56], and optimization techniques with crowdsourcing including CDAS [57], CrowdOP [36], CBD [58], task allocation [59], [60], knowledge base integration [61], etc. However, all the techniques above do not support the crowd skyline query with incomplete data. In addition, the work [62] combines Bayesian network and crowdsourcing to impute missing values, which is different from our goal of optimizing the query quality with crowdsourcing.

By contrast, the closest related work to ours is the crowd skyline query [23], [22]. As analyzed in Section 1, the work [22] is based on *unary* questions to impute missing values of objects, resulting in the inaccurate result. While in [23], they partition attributes into the observed attributes and the crowd ones, and assume that all values in crowd attributes are missing. Nevertheless, in real-life scenarios, it is easy to realize that, the values are usually missing over attributes randomly. Besides, both studies assume that data attributes are independent. It is worth noting that, our work studied in this paper allows attribute values missing randomly, and takes into account the data correlation.

Querying incomplete data. The study of incomplete data arises at the beginning of research due to its pervasiveness [63]. On the theoretical side, the foundational research from the 1980s, first by Imielinski and Lipski [25] and then by Abiteboul, Kanellakis, and Grahne [64] provide models of incompleteness appropriate for handling queries in different relational languages. In addition, some index structures have been presented, e.g., BR-tree, MOSAIC, bitmap, and VA file [65], [66], for organizing incomplete data. Also, several spatial queries over incomplete data have

1041-4347 (c) 2019 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TKDE.2019.2946798, IEEE Transactions on Knowledge and Data Engineering

been investigated, such as skyline queries [5], [6], [7], [8], [9], [10], [11], [12], [13], [14], ranking/top-k queries [15], [16], similarity queries [17], [18], [19], etc. Nevertheless, none of the aforementioned work adopts crowdsourcing techniques to process incomplete data. In addition, it is noteworthy that, the dominance relationship analysis (throughout the paper) is based on the traditional dominance relationship definition on *complete* data, instead of *incomplete* data. Hence, the techniques for skyline computation over incomplete data cannot be applied to our studied problem.

9 Conclusions

In this paper, we propose a novel crowd skyline query framework BayesCrowd. It takes into account the *data correlation*, and consists of two major phases, i.e., the modeling phase and the crowdsourcing maker phase. In the modeling phase, the query results are represented with *the c-table model*. We present an effective approach for c-table construction. Since probability computation in terms of conditions in the c-table is at least as hard as #SAT problem, we develop an ADPLL algorithm to accelerate computation. For the crowdsourcing phase, we put forward a suite of effective task selection strategies, which consider budget and latency constraints. We also introduce a *marginal utility function* to measure the benefit of crowdsourcing a task. Extensive experiments on both real and synthetic datasets demonstrate the superiority of BayesCrowd. In the future, we intend to further explore the quality optimization problem on answering incomplete data queries.

Acknowledgments. This work was supported in part by the National Key Research and Development Program of China under Grants No. 2018YFB1004003 and 2017YFB1400603, NSFC Grants No. 61972338, 61902343, 61825205, 61772459, and U1609217, National Science and Technology Major Project of China under Grant No. 50-D36B02-9002-16/19, the ZJU-Hikvision Joint Project, and the Fundamental Research Funds for the Central Universities. Yunjun Gao is the corresponding author of the work.

References

[1] S. Borzsonyi, D. Kossmann, and K. Stocker, "The skyline operator," in *ICDE*, pp. 421–430, 2001.
[2] D. Papadias, Y. Tao, G. Fu, and B. Seeger, "Progressive skyline computation in database systems," *ACM Trans. Database Syst.*, vol. 30, no. 1, pp. 41–82, 2005.
[3] X. Zhou, K. Li, Z. Yang, and K. Li, "Finding optimal skyline product combinations under price promotion," *IEEE Trans. Knowl. Data Eng.*, vol. 31, no. 1, pp. 138–151, 2018.
[4] X. Zhou, K. Li, Z. Yang, G. Xiao, and K. Li, "Progressive approaches for pareto optimal groups computation," *IEEE Trans. Knowl. Data Eng.*, vol. 31, no. 3, pp. 521–534, 2019.
[5] M. E. Khalefa, M. F. Mokbel, and J. J. Levandoski, "Skyline query processing for incomplete data," in *ICDE*, pp. 556–565, 2008.
[6] X. Miao, Y. Gao, B. Zheng, G. Chen, and H. Cui, "Top-k dominating queries on incomplete data," *IEEE Trans. Knowl. Data Eng.*, vol. 28, no. 1, pp. 252–266, 2016.
[7] Y. Wang, Z. Shi, J. Wang, L. Sun, and B. Song, "Skyline preference query based on massive and incomplete dataset," *IEEE Access*, vol. 5, pp. 3183–3192, 2017.
[8] A. A. Alwan, H. Ibrahim, N. I. Udzir, and F. Sidi, "An efficient approach for processing skyline queries in incomplete multidimensional database," *Arabian Journal for Science and Engineering*, vol. 41, no. 8, pp. 2927–2943, 2016.
[9] R. Bharuka and P. S. Kumar, "Finding skylines for incomplete data," in *ADC*, pp. 109–117, 2013.
[10] Y. Gulzar, A. A. Alwan, R. M. Abdullah, Q. Xin, and M. B. Swidan, "SCSA: Evaluating skyline queries in incomplete data," *Applied Intelligence*, vol. 49, no. 5, pp. 1636–1657, 2019.
[11] J. Lee, H. Im, and G.-w. You, "Optimizing skyline queries over incomplete data," *Inf. Sci.*, vol. 361, pp. 14–28, 2016.
[12] K. Zhang, H. Gao, X. Han, Z. Cai, and J. Li, "Probabilistic skyline on incomplete data," in *CIKM*, pp. 427–436, ACM, 2017.
[13] Y. Gulzar, A. A. Alwan, N. Salleh, and I. F. Al Shaikhli, "Processing skyline queries in incomplete database: Issues, challenges and future trends," *Journal of Computer Science*, vol. 13, no. 11, pp. 647–658, 2017.
[14] K. Zhang, H. Gao, X. Han, Z. Cai, and J. Li, "Modeling and computing probabilistic skyline on incomplete data," *IEEE Trans. Knowl. Data Eng.*, 2019.
[15] P. Haghani, S. Michel, and K. Aberer, "Evaluating top-k queries over incomplete data streams," in *CIKM*, pp. 877–886, 2009.
[16] M. A. Soliman, I. F. Ilyas, and S. Ben-David, "Supporting ranking queries on uncertain and incomplete data," *VLDB J.*, vol. 19, no. 4, pp. 477–501, 2010.
[17] W. Cheng, X. Jin, J.-T. Sun, X. Lin, X. Zhang, and W. Wang, "Searching dimension incomplete databases.," *IEEE Trans. Knowl. Data Eng.*, vol. 26, no. 3, pp. 725–738, 2014.
[18] A. Cuzzocrea and A. Nucita, "Enhancing accuracy and expressive power of range query answers over incomplete spatial databases via a novel reasoning approach," *Data Knowl. Eng.*, vol. 70, no. 8, pp. 702–716, 2011.
[19] X. Miao, Y. Gao, G. Chen, B. Zheng, and H. Cui, "Processing incomplete k nearest neighbor search," *IEEE Transactions on Fuzzy Systems*, vol. 24, no. 6, pp. 1349–1363, 2016.
[20] M. J. Franklin, D. Kossmann, T. Kraska, S. Ramesh, and R. Xin, "CrowdDB: Answering queries with crowdsourcing," in *SIGMOD*, pp. 61–72, 2011.
[21] M.-C. Yuen, I. King, and K.-S. Leung, "A survey of crowdsourcing systems," in *IEEE International Conference on Privacy, Security, Risk, and Trust, and IEEE International Conference on Social Computing*, pp. 766–773, IEEE, 2011.
[22] C. Lofi, K. El Maarry, and W.-T. Balke, "Skyline queries in crowd-enabled databases," in *EDBT*, pp. 465–476, 2013.
[23] J. Lee, D. Lee, and S.-W. Kim, "CrowdSky: Skyline computation with crowdsourcing.," in *EDBT*, pp. 125–136, 2016.
[24] R. J. Little and D. B. Rubin, *Statistical analysis with missing data, Second edition*. Wiley, 2002.
[25] T. Imieliński and W. Lipski, "Incomplete information in relational databases," *The Journal of ACM*, vol. 31, no. 4, pp. 761–791, 1984.
[26] J. Pei, B. Jiang, X. Lin, and Y. Yuan, "Probabilistic skylines on uncertain data," in *VLDB*, pp. 15–26, 2007.
[27] X. Zhou, K. Li, Y. Zhou, and K. Li, "Adaptive processing for distributed skyline queries over uncertain data," *IEEE Trans. Knowl. Data Eng.*, vol. 28, no. 2, pp. 371–384, 2016.
[28] X. Zhou, K. Li, G. Xiao, Y. Zhou, and K. Li, "Top k favorite probabilistic products queries," *IEEE Trans. Knowl. Data Eng.*, vol. 28, no. 10, pp. 2808–2821, 2016.
[29] L. Gondara and K. Wang, "Multiple imputation using deep denoising autoencoders," *arXiv preprint arXiv:1705.02737*, 2017.
[30] T. Rekatsinas, X. Chu, I. F. Ilyas, and C. Ré, "Holoclean: Holistic data repairs with probabilistic inference," *Proceedings of the VLDB Endowment*, vol. 10, no. 11, pp. 1190–1201, 2017.
[31] C. P. Gomes, A. Sabharwal, and B. Selman, "Model counting," 2008.
[32] T. Sang, P. Beame, and H. Kautz, "Heuristics for fast exact model counting," in *Theory and Applications of Satisfiability Testing*, pp. 226–240, Springer, 2005.
[33] C. Thiffault, F. Bacchus, and T. Walsh, "Solving non-clausal formulas with DPLL search," *Principles and Practice of Constraint Programming*, pp. 663–678, 2004.
[34] W. Wei and B. Selman, "A new approach to model counting," in *SAT*, pp. 324–339, Springer, 2005.
[35] A. Das Sarma, A. Parameswaran, H. Garcia-Molina, and A. Halevy, "Crowd-powered find algorithms," in *ICDE*, pp. 964–975, IEEE, 2014.
[36] J. Fan, M. Zhang, S. Kok, M. Lu, and B. C. Ooi, "CrowdOp: Query optimization for declarative crowdsourcing systems," *IEEE Trans. Knowl. Data Eng.*, vol. 27, no. 8, pp. 2078–2092, 2015.
[37] A. G. Parameswaran, H. Park, H. Garcia-Molina, N. Polyzotis, and J. Widom, "Deco: Declarative crowdsourcing," in *CIKM*, pp. 1203–1212, ACM, 2012.

1041-4347 (c) 2019 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TKDE.2019.2946798, IEEE Transactions on Knowledge and Data Engineering

[38] A. Marcus, E. Wu, D. R. Karger, S. Madden, and R. C. Miller, "Crowdsourced databases: Query processing with people," in *CIDR*, 2011.
[39] J. Gao, X. Liu, B. C. Ooi, H. Wang, and G. Chen, "An online cost sensitive decision-making method in crowdsourcing systems," in *SIGMOD*, pp. 217–228, 2013.
[40] A. G. Parameswaran, H. Garcia-Molina, H. Park, N. Polyzotis, A. Ramesh, and J. Widom, "CrowdScreen: Algorithms for filtering data with humans," in *SIGMOD*, pp. 361–372, ACM, 2012.
[41] B. Trushkowsky, T. Kraska, M. J. Franklin, and P. Sarkar, "Crowdsourced enumeration queries," in *ICDE*, pp. 673–684, IEEE, 2013.
[42] A. Marcus, E. Wu, D. Karger, S. Madden, and R. Miller, "Human-powered sorts and joins," *PVLDB*, vol. 5, no. 1, pp. 13–24, 2011.
[43] J. Wang, T. Kraska, M. J. Franklin, and J. Feng, "CrowdER: Crowdsourcing entity resolution," *PVLDB*, vol. 5, no. 11, pp. 1483–1494, 2012.
[44] J. Wang, G. Li, T. Kraska, M. J. Franklin, and J. Feng, "Leveraging transitive relations for crowdsourced joins," in *SIGMOD*, pp. 229–240, ACM, 2013.
[45] S. Wang, X. Xiao, and C.-H. Lee, "Crowd-based deduplication: An adaptive approach," in *SIGMOD*, pp. 1263–1277, ACM, 2015.
[46] S. B. Davidson, S. Khanna, T. Milo, and S. Roy, "Using the crowd for top-k and group-by queries," in *ICDT*, pp. 225–236, ACM, 2013.
[47] S. Guo, A. Parameswaran, and H. Garcia-Molina, "So who won?: Dynamic max discovery with the crowd," in *SIGMOD*, pp. 385–396, 2012.
[48] P. Venetis, H. Garcia-Molina, K. Huang, and N. Polyzotis, "Max algorithms in crowdsourcing environments," in *WWW*, pp. 989–998, ACM, 2012.
[49] V. Verroios, P. Lofgren, and H. Garcia-Molina, "tDP: An optimal-latency budget allocation strategy for crowdsourced maximum operations," in *SIGMOD*, pp. 1047–1062, ACM, 2015.
[50] E. Ciceri, P. Fraternali, D. Martinenghi, and M. Tagliasacchi, "Crowdsourcing for top-k query processing over uncertain data," *IEEE Trans. Knowl. Data Eng.*, vol. 28, no. 1, pp. 41–53, 2016.
[51] L. de Alfaro, V. Polychronopoulos, and N. Polyzotis, "Efficient techniques for crowdsourced top-k lists," in *Fourth AAAI Conference on Human Computation and Crowdsourcing*, 2016.
[52] J. Lee, D. Lee, and S.-w. Hwang, "CrowdK: Answering top-k queries with crowdsourcing," *Inf. Sci.*, vol. 399, pp. 98–120, 2017.
[53] Y. Li, N. M. Kou, H. Wang, Z. Gong, *et al.*, "A confidence-aware top-k query processing toolkit on crowdsourcing," *PVLDB*, vol. 10, no. 12, pp. 1909–1912, 2017.
[54] X. Zhang, G. Li, and J. Feng, "Crowdsourced top-k algorithms: An experimental evaluation," *PVLDB*, vol. 9, no. 8, pp. 612–623, 2016.
[55] G. Li, J. Wang, Y. Zheng, and M. Franklin, "Crowdsourced data management: A survey," *IEEE Trans. Knowl. Data Eng.*, vol. 28, no. 9, pp. 2296–2319, 2016.
[56] G. Li, Y. Zheng, J. Fan, J. Wang, and R. Cheng, "Crowdsourced data management: Overview and challenges," in *SIGMOD*, pp. 1711–1716, ACM, 2017.
[57] X. Liu, M. Lu, B. C. Ooi, Y. Shen, S. Wu, and M. Zhang, "CDAS: A crowdsourcing data analytics system," *PVLDB*, vol. 5, no. 10, pp. 1040–1051, 2012.
[58] G. Li, C. Chai, J. Fan, X. Weng, J. Li, Y. Zheng, Y. Li, X. Yu, X. Zhang, and H. Yuan, "CDB: Optimizing queries with crowd-based selections and joins," in *SIGMOD*, pp. 1463–1478, ACM, 2017.
[59] Y. Tong, L. Chen, Z. Zhou, H. V. Jagadish, L. Shou, and W. Lv, "SLADE: A smart large-scale task decomposer in crowdsourcing," *IEEE Trans. Knowl. Data Eng.*, vol. 30, no. 8, pp. 1588–1601, 2018.
[60] Y. Tong, J. She, B. Ding, L. Wang, and L. Chen, "Online mobile micro-task allocation in spatial crowdsourcing," in *ICDE*, pp. 49–60, IEEE, 2016.
[61] R. Meng, L. Chen, Y. Tong, and C. Zhang, "Knowledge base semantic integration using crowdsourcing," *IEEE Trans. Knowl. Data Eng.*, vol. 29, no. 5, pp. 1087–1100, 2017.
[62] C. Ye, H. Wang, J. Li, H. Gao, and S. Cheng, "Crowdsourcing-enhanced missing values imputation based on bayesian network," in *DASFAA*, pp. 67–81, 2016.
[63] J. Graham, *Missing data: Analysis and design*. Statistics for Social and Behavioral Sciences, Springer, 2012.
[64] S. Abiteboul, P. Kanellakis, and G. Grahne, "On the representation and querying of sets of possible worlds," *Theoretical Computer Science*, vol. 78, no. 1, pp. 159–187, 1991.
[65] G. Canahuate, M. Gibas, and H. Ferhatosmanoglu, "Indexing incomplete databases," in *EDBT*, pp. 884–901, 2006.
[66] B. C. Ooi, C. H. Goh, and K.-L. Tan, "Fast high-dimensional data search in incomplete databases," in *PVLDB*, pp. 357–367, 1998.

Xiaoye Miao received the PhD degree in computer science from Zhejiang University, China, in 2017. She is currently an assistant professor in the Center for Data Science, Zhejiang University, China. She was ever a research/postdoctoral fellow at University of New South Wales and City University of Hong Kong, respectively. Her research interests include uncertain/incomplete databases, data pricing, data cleaning, and graph management. She is a member of the IEEE and the CCF.

Yunjun Gao received the PhD degree in computer science from Zhejiang University, China, in 2008. He is currently a professor in the College of Computer Science, Zhejiang University, China. His research interests include database, big data management and analytics, and AI interaction with DB technology. He is a member of the ACM and the IEEE, and a senior member of the CCF.

Su Guo received the BS degree in computer science from Xidian University, China, in 2016. She is currently working toward the MS degree in the College of Computer Science, Zhejiang University, China. Her research interests include incomplete databases and query processing.

Lu Chen received the PhD degree in computer science from Zhejiang University, China, in 2016. Prior to joining Aalborg University in 2017, she was a postdoctoral fellow in the School of Computer Science and Engineering, Nanyang Technological University, Singapore. She is currently an associate professor in Aalborg University, Denmark. Her research interests include metric data management and big data analytics.

Jianwei Yin received the PhD degree in computer science from Zhejiang University, in 2001. He is currently a professor in the College of Computer Science, Zhejiang University. He is a visiting scholar at the Georgia Institute of Technology, in 2008. His research interests include service computing and data management.

Qing Li is a chair professor of Data Science, the Hong Kong Polytechnic University since Dec 2018. Prior to that, he has taught at City University of Hong Kong, the Hong Kong University of Science and Technology, and the Australian National University (Canberra, Australia). He is currently a Fellow of IET (formerly IEE), a Senior Member of IEEE, a member of ACM-SIGMOD and IEEE Technical Committee on Data Engineering. He is the Chairperson of the Hong Kong Web Society, and also served/is serving as an executive committee (EXCO) member of IEEE-Hong Kong Computer Chapter and ACM Hong Kong Chapter.

1041-4347 (c) 2019 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.